Practical No.1

Aim: Simple Linear Regression

```
import pandas as pd
df = pd.read_csv('/content/Study.csv')
df
```

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30
5	1.5	20

```
x = df.iloc[:,0].values
x
```

```
array([2.5, 5.1, 3.2, 8.5, 3.5, 1.5, 9.2, 5.5, 8.3, 2.7, 7.7, 5.9, 4.5,
       3.3, 1.1, 8.9, 2.5, 1.9, 6.1, 7.4, 2.7, 4.8, 3.8, 6.9, 7.8])
```

```
y = df.iloc[:,1]
y
```

	Scores
0	21
1	47
2	27
3	75
4	30
5	20

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=10)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((20,), (5,), (20,), (5,))
```

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x_train.reshape(-1,1), y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
regressor_pred = regressor.predict(x_test.reshape(-1,1))
```

```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test,regressor_pred)
```

```
5.632881746692994
```

Practical No.2

Aim: Multiple Linear Regression

```
import pandas as pd
df=pd.read_csv('/content/Advertising.csv')
df.head(5)
```

	ID	TV	Radio	Newspaper	Sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   ID         200 non-null    int64  
 1   TV         200 non-null    float64
 2   Radio      200 non-null    float64
 3   Newspaper  200 non-null    float64
 4   Sales      200 non-null    float64
dtypes: float64(4), int64(1)
memory usage: 7.9 KB
```

```
df.drop('ID',axis=1,inplace=True)
df.head(2)
```

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   TV         200 non-null    float64
 1   Radio      200 non-null    float64
 2   Newspaper  200 non-null    float64
 3   Sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
```

```
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
x=df.iloc[:,:-1]
x.shape
```

```
(200, 3)
```

```
y=df.iloc[:,-1]
y.shape
```

```
(200,)
```

from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.25,random_state=1)
xtrain.shape, xtest.shape, ytrain.shape, ytest.shape

```
((150, 3), (50, 3), (150,), (50,))
```

regressor.fit(xtrain,ytrain)

```
▾ LinearRegression
LinearRegression()
```

predictions=regressor.predict(xtest)
results=pd.DataFrame({'Actual':ytest,'Predicted':predictions})
results

	Actual	Predicted
58	23.8	21.709103
40	16.6	16.410552
34	9.5	7.609551
102	14.8	17.807696
184	17.6	18.614636
198	25.5	23.835740
95	16.9	16.324887

from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
mean_absolute_error(ytest,predictions)

```
1.0668917082595213
```

mean_squared_error(ytest,predictions)

```
1.9730456202283373
```

r2_score(ytest,predictions)

```
0.9156213613792232
```

regressor.intercept_

```
np.float64(2.87696662231793)
```

regressor.coef_

```
array([0.04656457, 0.17915812, 0.00345046])
```

Practical No.3

Aim: Logistic Regression

```
import pandas as pd
df=pd.read_csv('/content/pima-diabetes.csv')
df.shape
```

```
(768, 9)
```

```
df.columns
```

```
Index(['Pregnancies', 'Weight', 'BP1', 'BP2', 'Insulin', 'Skin', 'BMI', 'Age',
       'Diabetes'],
      dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Pregnancies  768 non-null    int64
 1   Weight       768 non-null    int64
 2   BP1          768 non-null    int64
 3   BP2          768 non-null    int64
 4   Insulin      768 non-null    int64
 5   Skin         768 non-null    float64
 6   BMI          768 non-null    float64
 7   Age          768 non-null    int64
 8   Diabetes     768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
df['Diabetes'].unique()
```

```
array([1, 0])
```

```
df['Diabetes'].value_counts()
```

	count
Diabetes	
0	500
1	268

dtype: int64

```
df.describe()
```

	Pregnancies	Weight	BP1	BP2	Insulin	Skin	BMI	Age	Diabetes
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

df.head()

	Pregnancies	Weight	BP1	BP2	Insulin	Skin	BMI	Age	Diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

df.tail()

	Pregnancies	Weight	BP1	BP2	Insulin	Skin	BMI	Age	Diabetes
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

df.sample(5)

	Pregnancies	Weight	BP1	BP2	Insulin	Skin	BMI	Age	Diabetes
716	3	173	78	39	185	33.8	0.970	31	1
664	6	115	60	39	0	33.7	0.245	40	1
743	9	140	94	0	0	32.7	0.734	45	1
559	11	85	74	0	0	30.1	0.300	35	0
408	8	197	74	0	0	25.9	1.191	39	1

x=df.iloc[:,:-1]

x

	Pregnancies	Weight	BP1	BP2	Insulin	Skin	BMI	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	0	0	30.1	0.349	47
767	1	93	70	31	0	30.4	0.315	23

768 rows × 8 columns

y=df.iloc[:,-1]

from sklearn.preprocessing import StandardScaler

```
ss=StandardScaler()
x_scaled=ss.fit_transform(x)
x_scaled
```

```
array([[ 0.63994726,  0.84832379,  0.14964075, ...,  0.20401277,
         0.46849198,  1.4259954 ],
       [-0.84488505, -1.12339636, -0.16054575, ..., -0.68442195,
        -0.36506078, -0.19067191],
       [ 1.23388019,  1.94372388, -0.26394125, ..., -1.10325546,
         0.60439732, -0.10558415],
       ...,
       [ 0.3429808 ,  0.00330087,  0.14964075, ..., -0.73518964,
        -0.68519336, -0.27575966],
       [-0.84488505,  0.1597866 , -0.47073225, ..., -0.24020459,
        -0.37110101,  1.17073215],
       [-0.84488505, -0.8730192 ,  0.04624525, ..., -0.20212881,
        -0.47378505, -0.87137393]])
```

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x_scaled,y,test_size=0.25,random_state=1)
```

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(xtrain,ytrain)
```

```
LogisticRegression
LogisticRegression()
```

```
predictions=lr.predict(xtest)
xtrain.shape,xtest.shape
```

```
((576, 8), (192, 8))
```

```
predictions
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0])
```

```
from sklearn.metrics import accuracy_score
accuracy_score(ytest,predictions)
```

```
0.7760416666666666
```

Practical No.4

Aim: Confusion Matrix

```
import pandas as pd
data ={'y_actual':[1,0,0,1,0,1,0,0,1,0,1,0],'y_pred':[1,1,0,1,0,1,1,0,1,0,0,0]}
df= pd.DataFrame(data,columns=['y_actual','y_pred'])
df
```

	y_actual	y_pred
0	1	1
1	0	1
2	0	0
3	1	1
4	0	0
5	1	1
6	0	1
7	0	0
8	1	1
9	0	0
10	1	0
11	0	0

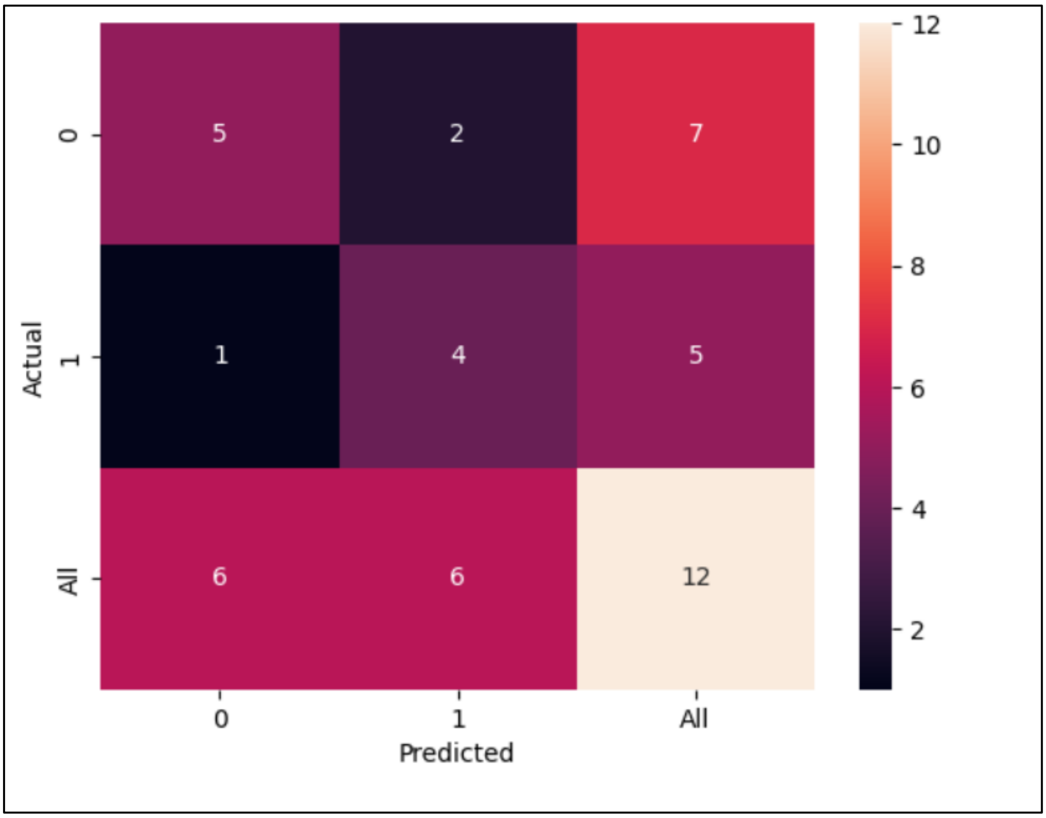
```
confusion_matrix=pd.crosstab(df['y_actual'],df['y_pred'],rownames=['Actual'],colnames=['Pred
cted'])
confusion_matrix
```

Predicted	0	1
Actual		
0	5	2
1	1	4

```
import seaborn as sns
import matplotlib.pyplot as plt
confusion_matrix=pd.crosstab(df['y_actual'],df['y_pred'],rownames=['Actual'],colnames=['Predi
cted'],margins=True)
confusion_matrix
```

Predicted	0	1	All
Actual			
0	5	2	7
1	1	4	5
All	6	6	12

```
sns.heatmap(confusion_matrix,annot=True)
plt.show()
```



confusion_matrix = pd.crosstab(df['y_actual'], df['y_pred'], rownames = ['Actual'], colnames = ['Predicted'], margins = True)
confusion_matrix

Predicted	0	1	All
Actual			
0	5	2	7
1	1	4	5
All	6	6	12

Practical No.5

Aim: Feature selection using Lasso Regression.

```
import pandas as pd
df = pd.read_csv("/content/Boston_Housing.xls - Data (1).csv")
df
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88	11.9

506 rows × 14 columns

```
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_sc=sc.fit_transform(x)
x_sc
```

```
array([[-0.41978194,  0.28482986, -1.2879095 , ..., -1.45900038,
         0.44105193, -1.0755623 ],
       [-0.41733926, -0.48772236, -0.59338101, ..., -0.30309415,
         0.44105193, -0.49243937],
       [-0.41734159, -0.48772236, -0.59338101, ..., -0.30309415,
         0.39642699, -1.2087274 ],
       ...,
       [-0.41344658, -0.48772236,  0.11573841, ...,  1.17646583,
         0.44105193, -0.98304761],
       [-0.40776407, -0.48772236,  0.11573841, ...,  1.17646583,
         0.4032249 , -0.86530163],
       [-0.41500016, -0.48772236,  0.11573841, ...,  1.17646583,
         0.44105193, -0.66905833]])
```

```
from sklearn.linear_model import Lasso
names = x.columns

def lasso(alphas):
 df1=pd.DataFrame()
 df1['FeatureName']=names
 for alpha in alphas:
  lasso=Lasso(alpha=alpha)
  lasso.fit(x_sc,y)
```

```
  column_name='Alpha=%f' %alpha
  df1[column_name]=lasso.coef_
 return df1
```

```
lasso([0.0001,0.001,0.01,0.1,0.5,1,10,100])
```

	FeatureName	Alpha=0.000100	Alpha=0.001000	Alpha=0.010000	Alpha=0.100000	Alpha=0.500000	Alpha=1.000000	Alpha=10.000000	Alpha=100.000000
0	CRIM	-0.927866	-0.925348	-0.900245	-0.632304	-0.115265	-0.000000	-0.0	-0.0
1	ZN	1.081086	1.076739	1.035916	0.708409	0.000000	0.000000	0.0	0.0
2	INDUS	0.139960	0.131471	0.046924	-0.000000	-0.000000	-0.000000	-0.0	-0.0
3	CHAS	0.681771	0.682060	0.684152	0.657607	0.397079	0.000000	0.0	0.0
4	NOX	-2.055877	-2.048349	-1.980551	-1.574193	-0.000000	-0.000000	-0.0	-0.0
5	RM	2.674402	2.675950	2.687312	2.826269	2.974259	2.713355	0.0	0.0
6	AGE	0.019026	0.015049	0.000000	-0.000000	-0.000000	-0.000000	-0.0	-0.0
7	DIS	-3.103667	-3.100300	-3.058301	-2.422079	-0.170569	-0.000000	0.0	0.0
8	RAD	2.660381	2.643836	2.481844	1.195937	-0.000000	-0.000000	-0.0	-0.0
9	TAX	-2.074993	-2.058853	-1.899442	-0.846468	-0.000000	-0.000000	-0.0	-0.0
10	PTRATIO	-2.060372	-2.058263	-2.038645	-1.922493	-1.598449	-1.343549	-0.0	-0.0
11	B	0.849183	0.848414	0.839724	0.762165	0.543139	0.180957	0.0	0.0
12	LSTAT	-3.743418	-3.741514	-3.730874	-3.726184	-3.666144	-3.543381	-0.0	-0.0

Practical No.6

Aim: Hyper parameter tunning using Ridge Regression.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
import matplotlib.pyplot as plt

# Load your dataset
df = pd.read_csv("/content/Boston_Housing.xls - Data (1).csv")

# Split into features and target
x = df.iloc[:, :-1]
y = df.iloc[:, -1]
feature_names = x.columns

# Scale the features
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)

# Ridge regression for different alpha values
def ridge(alphas):
  results = pd.DataFrame()
  results['FeatureName'] = feature_names
  for alpha in alphas:
    ridge_model = Ridge(alpha=alpha)
    ridge_model.fit(x_scaled, y)
    results[f'Alpha={alpha}'] = ridge_model.coef_
  return results

# Run the function
ridge_df = ridge([0.0001, 0.001, 0.01, 0.1, 1, 10, 100])
print(ridge_df)
```

```
   FeatureName  Alpha=0.0001  Alpha=0.001  Alpha=0.01  Alpha=0.1   Alpha=1  \
0         CRIM     -0.928145    -0.928138   -0.928061  -0.927301 -0.919871
1           ZN      1.081567     1.081553    1.081414   1.080027  1.066461
2        INDUS      0.140898     0.140876    0.140657   0.138479  0.117385
3         CHAS      0.681740     0.681743    0.681775   0.682089  0.685127
4          NOX     -2.056715    -2.056690   -2.056437  -2.053909 -2.029010
5           RM      2.674231     2.674239    2.674318   2.675104  2.682754
6          AGE      0.019465     0.019460    0.019401   0.018821  0.013158
7          DIS     -3.104042    -3.104017   -3.103775  -3.101353 -3.077340
8          RAD      2.662210     2.662145    2.661490   2.654964  2.591538
9          TAX     -2.076775    -2.076713   -2.076098  -2.069961 -2.010558
10     PTRATIO     -2.060606    -2.060598   -2.060523  -2.059773 -2.052385
11           B      0.849268     0.849268    0.849264   0.849227  0.848848
12       LSTAT     -3.743626    -3.743614   -3.743496  -3.742319 -3.730666

    Alpha=10  Alpha=100
0  -0.859051  -0.652004
1   0.954975   0.578885
2  -0.041327  -0.402318
3   0.707780   0.739944
4  -1.812611  -0.925045
5   2.742344   2.777933
6  -0.032383  -0.172802
7  -2.856756  -1.688537
8   2.097823   0.699906
9  -1.565395  -0.608373
10 -1.987751  -1.661424
11  0.844709   0.778625
12 -3.623942  -2.961415
```

Practical No.7

Aim: Decision Tree Classifier

```
from sklearn.datasets import load_iris
ds = load_iris()
x=ds.data
y=ds.target
ds.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
x
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
ds.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=42)
xtrain.shape,xtest.shape,ytrain.shape,ytest.shape
```

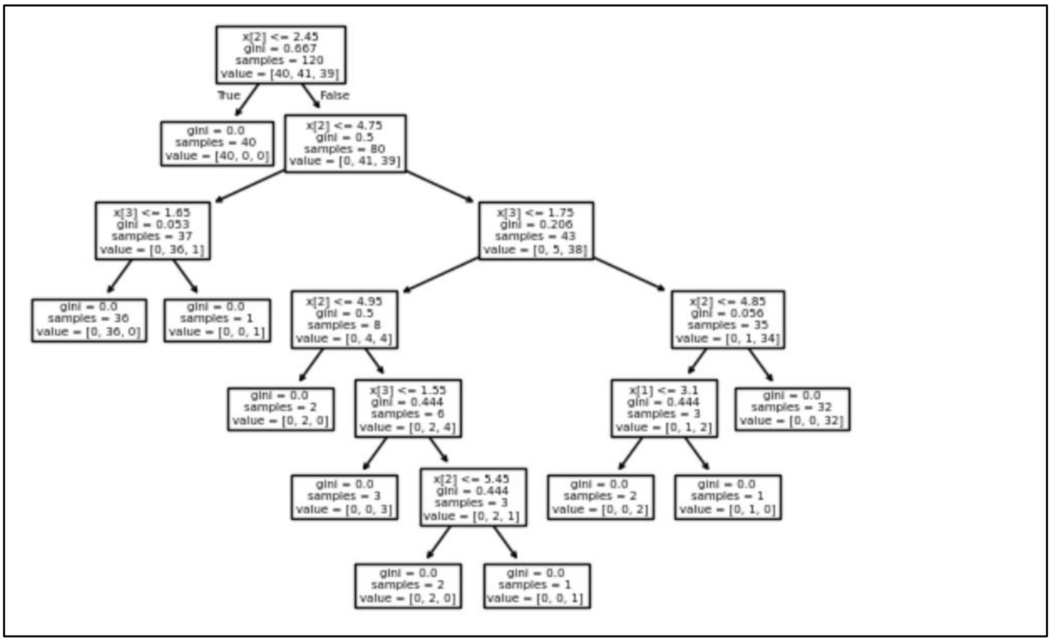
```
((120, 4), (30, 4), (120,), (30,))
```

from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(xtrain,ytrain)

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

from sklearn import tree
tree.plot_tree(dt)

```
[Text(0.3076923076923077, 0.9285714285714286, 'x[2] <= 2.45\ngini = 0.667\nsamples = 120\nvalue = [40, 41, 39]'),
 Text(0.23076923076923078, 0.7857142857142857, 'gini = 0.0\nsamples = 40\nvalue = [40, 0, 0]'),
 Text(0.2692307692307693, 0.8571428571428572, 'True  '),
 Text(0.38461538461538464, 0.7857142857142857, 'x[2] <= 4.75\ngini = 0.5\nsamples = 80\nvalue = [0, 41, 39]'),
 Text(0.34615384615384615, 0.8571428571428572, '  False'),
 Text(0.15384615384615385, 0.6428571428571429, 'x[3] <= 1.65\ngini = 0.053\nsamples = 37\nvalue = [0, 36, 1]'),
 Text(0.07692307692307693, 0.5, 'gini = 0.0\nsamples = 36\nvalue = [0, 36, 0]'),
 Text(0.23076923076923078, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.6153846153846154, 0.6428571428571429, 'x[3] <= 1.75\ngini = 0.206\nsamples = 43\nvalue = [0, 5, 38]'),
 Text(0.38461538461538464, 0.5, 'x[2] <= 4.95\ngini = 0.5\nsamples = 8\nvalue = [0, 4, 4]'),
 Text(0.3076923076923077, 0.35714285714285715, 'gini = 0.0\nsamples = 2\nvalue = [0, 2, 0]'),
 Text(0.46153846153846156, 0.35714285714285715, 'x[3] <= 1.55\ngini = 0.444\nsamples = 6\nvalue = [0, 2, 4]'),
 Text(0.38461538461538464, 0.21428571428571427, 'gini = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(0.5384615384615384, 0.21428571428571427, 'x[2] <= 5.45\ngini = 0.444\nsamples = 3\nvalue = [0, 2, 1]'),
 Text(0.46153846153846156, 0.07142857142857142, 'gini = 0.0\nsamples = 2\nvalue = [0, 2, 0]'),
 Text(0.6153846153846154, 0.07142857142857142, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.8461538461538461, 0.5, 'x[2] <= 4.85\ngini = 0.056\nsamples = 35\nvalue = [0, 1, 34]'),
 Text(0.7692307692307693, 0.35714285714285715, 'x[1] <= 3.1\ngini = 0.444\nsamples = 3\nvalue = [0, 1, 2]'),
 Text(0.6923076923076923, 0.21428571428571427, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
 Text(0.8461538461538461, 0.21428571428571427, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.9230769230769231, 0.35714285714285715, 'gini = 0.0\nsamples = 32\nvalue = [0, 0, 32]')]
```



from sklearn.metrics import accuracy_score,confusion_matrix
predictions=dt.predict(xtest)
accuracy_score(ytest,predictions)

```
1.0
```

confusion_matrix(ytest,predictions)

```
array([[10,  0,  0],
       [ 0,  9,  0],
       [ 0,  0, 11]])
```

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=50)
rf.fit(xtrain,ytrain)
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=50)
```

```
predictions2=rf.predict(xtest)
accuracy_score(ytest,predictions2)
```

```
1.0
```

Practical No.8

Aim: Support Vector Machine (SVM)

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
cancer=load_breast_cancer()
x,y=cancer.data,cancer.target
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=42)
kernels=['linear','poly','rbf']
kernels
```

```
['linear', 'poly', 'rbf']
```

```
best_kernel=None
best_accuracy=0
for kernel in kernels:
  model=SVC(kernel=kernel)
  model.fit(xtrain,ytrain)
  ypred=model.predict(xtest)
  accuracy=accuracy_score(ytest,ypred)
  print('For ',kernel,' accuracy is ',accuracy)
  if accuracy>best_accuracy:
    best_accuracy=accuracy
    best_kernel=kernel
```

```
For  linear  accuracy is  0.956140350877193
For  poly  accuracy is  0.9473684210526315
For  rbf  accuracy is  0.9473684210526315
```

```
print('For the given problem best kernel is ',best_kernel,' with accuracy ',best_accuracy)
```

```
For the given problem best kernel is  linear  with accuracy  0.956140350877193
```

Practical No.9

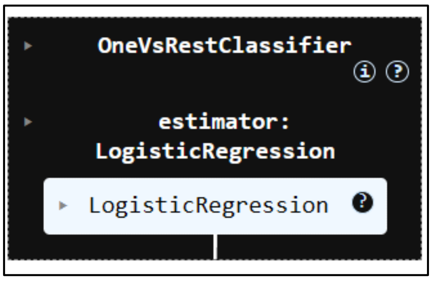
Aim: Implementation of binary classifier using One vs One and One vs Rest scheme.

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier
from sklearn.metrics import accuracy_score
iris =load_iris()
X=iris.data
Y=iris.target
xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.25,random_state=42)
ovo=OneVsOneClassifier(LogisticRegression(max_iter=200))
ovo.fit(xtrain,ytrain)
```


```
pred_ovo=ovo.predict(xtest)
print("Accuracy with one versus one approach is",accuracy_score(ytest,pred_ovo))
```

```
Accuracy with one versus one approach is 1.0
```

```
ovr=OneVsRestClassifier(LogisticRegression(max_iter=200))
ovr.fit(xtrain,ytrain)
```




```
pred_ovr=ovr.predict(xtest)
print("Accuracy with one versus rest approach is",accuracy_score(ytest,pred_ovr))
```

```
Accuracy with one versus rest approach is 0.9736842105263158
```